模組建構 **def**：

A. 模擬馬可夫鏈收斂

B.十進位轉二進位

C. 找所有質因數

D. 矩陣乘法

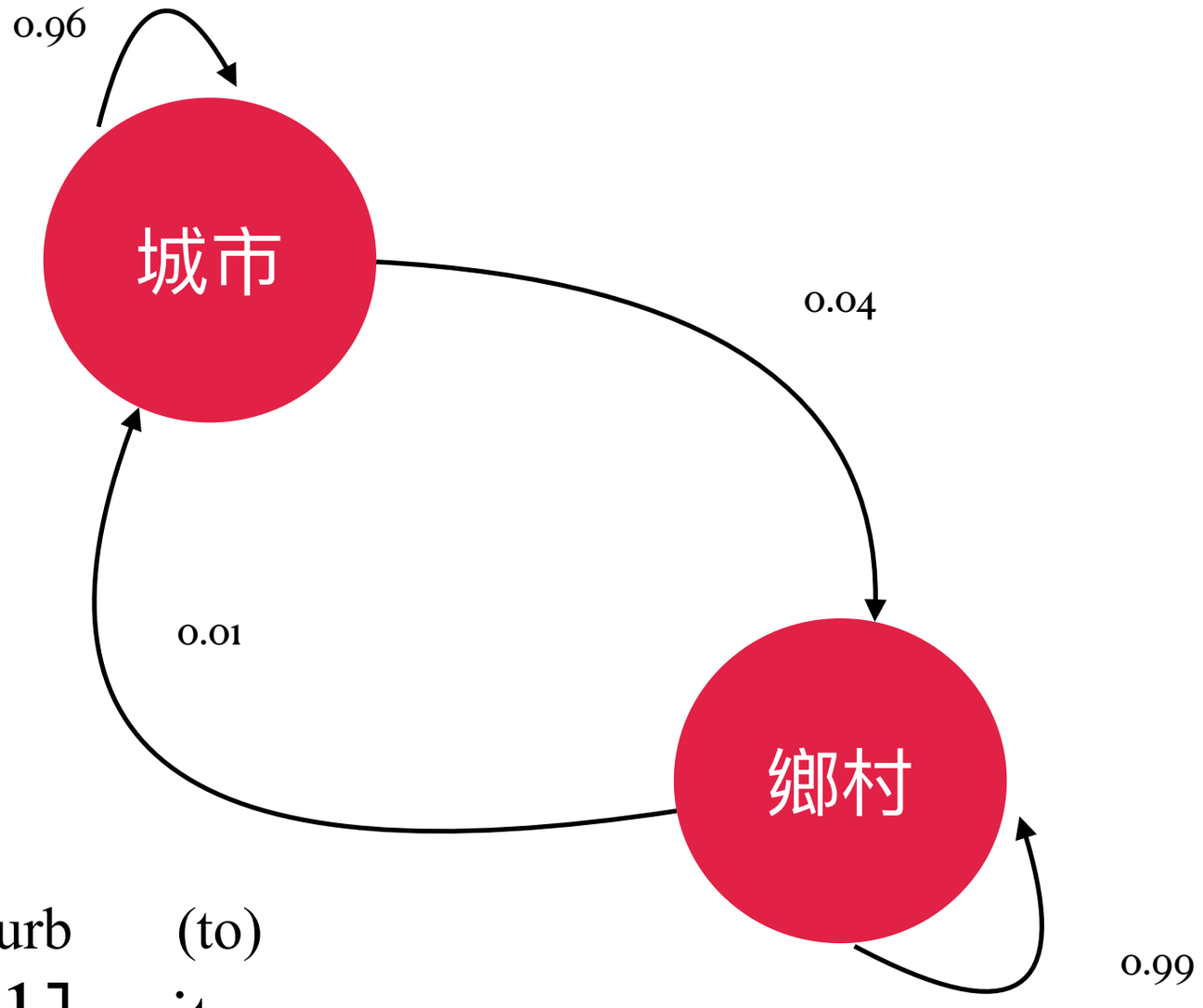
E. 數獨檢查

$$\begin{bmatrix} 0.96 & 0.01 \\ 0.04 & 0.99 \end{bmatrix} \begin{bmatrix} 80 \\ 175 \end{bmatrix}$$

$$Px_0 \rightarrow x_1$$

$$P = \begin{array}{c} \begin{matrix} \text{city} & \text{suburb} \end{matrix} \quad \text{(from)} \\ \begin{bmatrix} 0.96 & 0.01 \\ 0.04 & 0.99 \end{bmatrix} \end{array} \begin{array}{l} \text{(to)} \\ \text{city} \\ \text{suburb} \end{array}$$

馬可夫鏈的移轉機率
Markov chain



$$P = \begin{array}{c} \text{(from)} \\ \begin{matrix} \text{city} & \text{suburb} \end{matrix} \\ \begin{bmatrix} 0.96 & 0.01 \\ 0.04 & 0.99 \end{bmatrix} \end{array} \begin{array}{c} \text{(to)} \\ \text{city} \\ \text{suburb} \end{array}$$

長期預測:是否會收斂？

$$x_0 \to x_1 \to \ldots \to x_n \to x_{n+1} \ldots$$

$$Px_n \to x_{n+1}$$

總會有一個n，使得
x_n和x_{n+1}都變成x

長期預測:如果收斂

$$x_0 \to x_1 \to \ldots \to x \to x$$

$$Px_n \to x_{n+1}$$

$$Px = x$$

總會有一個n，使得
x_n和x_{n+1}都變成x

長期預測:如果收斂

$$x_0 \to x_1 \to \ldots \to x \to x$$

$$Px_n \to x_{n+1}$$

$$\|Px - x\| < 10^{-6}$$

向量長度 norm

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\|x\| = \sqrt{x_1^2 + x_2^2}$$

```python
import numpy as np
from numpy.linalg import norm

x = np.array([[80],[175]])
print(norm(x))
print(np.sqrt(x[0]*x[0]+x[1]*x[1]))
```

$$\|x\| = \sqrt{x_1^2 + x_2^2}$$

```
192.41881404893857
[192.41881405]
```

設定序列收斂條件

$$\|Px - x\| < 10^{-6}$$

$$\|Px - x\| < \epsilon$$

$$\|Px - x\| < 10^{-6}$$

```
import numpy as np
from numpy.linalg import norm

P = np.array([[0.96, 0.01],[0.04,0.99]])
x = np.array([[80],[175]])
print(norm(P@x -x)< pow(10,-6))
```

收斂條件，不成立

```
False
```

使用while迴圈模擬

```
while not norm(P@x-x) < pow(10,-6):
    x = P@x
```

更新狀態

收斂條件不成立

```python
import numpy as np
from numpy.linalg import norm

P = np.array([[0.96, 0.01],[0.04,0.99]])
x = np.array([[80],[175]])

while not norm(P@x-x) < pow(10,-6):
    x = P@x
print(x)
```

```
[[ 51.00001367]
 [203.99998633]]
```

```python
import numpy as np
from numpy.linalg import norm

P = np.array([[0.96, 0.01],[0.04,0.99]])
x = np.array([[80],[175]])
loop_max = 1000
loop = 0
while not norm(P@x-x) < pow(10,-6):
    x = P@x
    loop += 1
    if loop > loop_max:
        break
print(x)
print(loop)
```

增加迴圈數
如果迴圈數超過最大迴圈，中斷迴圈執行

```
[[ 51.00001367]
 [203.99998633]]
```

```python
import numpy as np
from numpy.linalg import norm

P = np.array([[0.96, 0.01],[0.04,0.99]])
x = np.array([[80],[175]])
loop_max = 1000
loop = 0
while not norm(P@x-x) < pow(10,-6):
    x = P@x
    loop += 1
    if loop > loop_max:
        break
print(x)
print(loop)
```

增加迴圈數
如果迴圈數超過最大迴圈，中斷迴圈執行

```
[[ 51.00001367]
 [203.99998633]]
```

模組化

markov

markov.py

自己建構馬可夫模組

```
import numpy as np
from numpy.linalg import norm
def sim(P,x):
loop_max = 1000
loop = 0
while not norm(P@x-x) < pow(10,-6):
    x = P@x
    loop += 1
    if loop > loop_max:
        break
print(x)
```

定義sim

輸入參數包含：P, x

函數或方法
sim

markov.py

```python
import numpy as np
from numpy.linalg import norm
def sim(P,x):
loop_max = 1000
loop = 0
while not norm(P@x-x) < pow(10,-6):
    x = P@x
    loop += 1
    if loop > loop_max:
        break
print(x)
```

demo_markov.py

```python
import numpy as np
from markov import sim

P = np.array([[0.96, 0.01],[0.04,0.99]])
x = np.array([[80],[175]])
sim(P,x)
```

從markov模組匯入sim

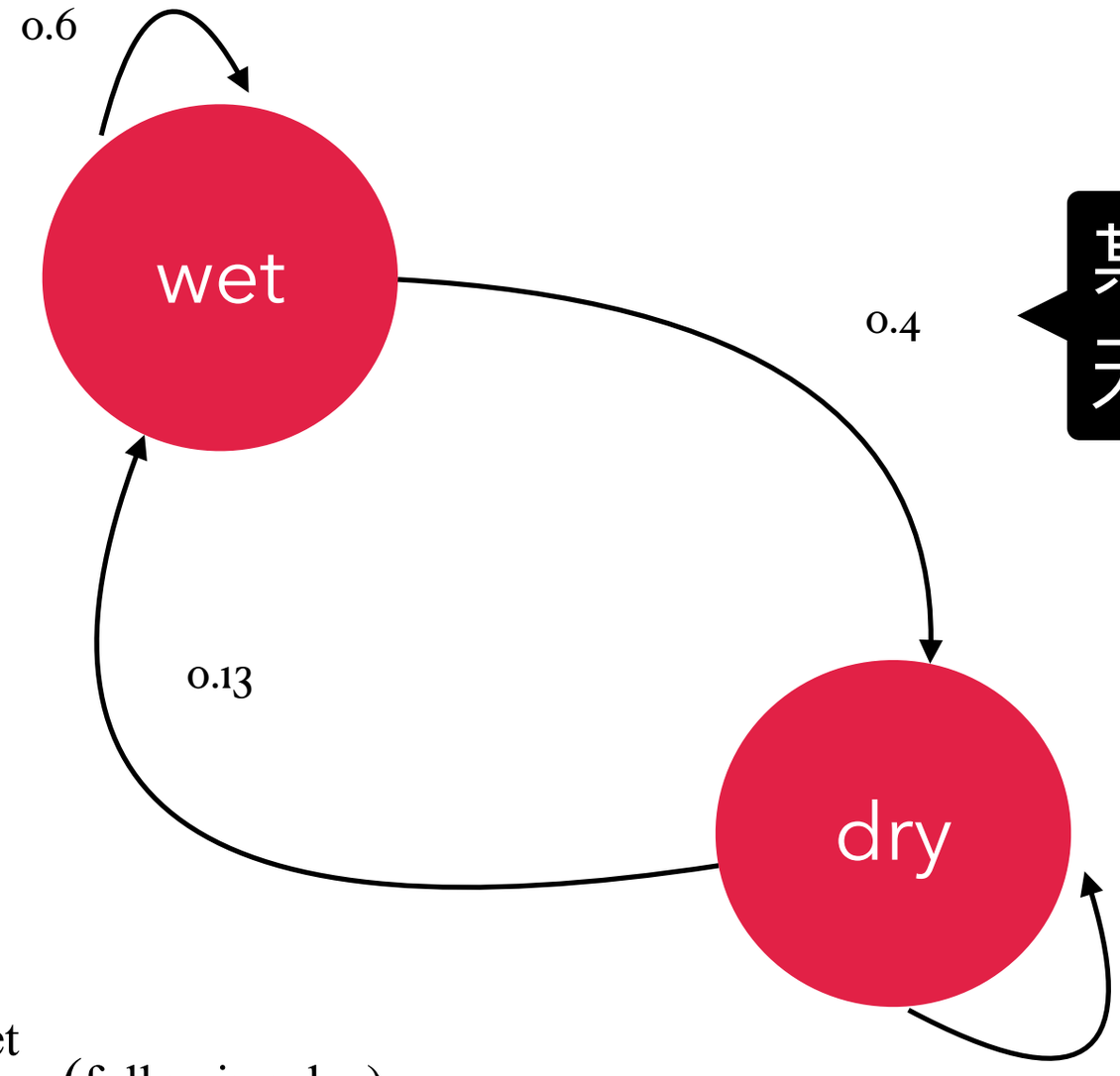
使用sim

```
[[ 51.00001367]
 [203.99998633]]
```

馬可夫sim模組可以應用的天氣預測問題

馬可夫鏈的移轉機率
Markov chain

$$x_0 = \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix}$$



某一天下雨，隔天好天氣的機率

$$P = \begin{array}{c} \text{(A given day)} \\ \text{wet} \quad \text{dry} \end{array} \begin{bmatrix} 0.6 & 0.13 \\ 0.4 & 0.87 \end{bmatrix} \begin{array}{l} \text{wet} \\ \text{dry} \end{array} \text{(following day)}$$

某一天好天氣，隔天好天氣的機率

$$P = \begin{matrix} \text{(A given day)} \\ \text{wet} \quad \text{dry} \end{matrix} \begin{bmatrix} 0.6 & 0.13 \\ 0.4 & 0.87 \end{bmatrix} \begin{matrix} \text{wet} \\ \text{dry} \end{matrix} \quad \text{(following day)}$$

```
import numpy as np
from markov import sim

Q = np.array([[0.6, 0.13],[0.4,0.87]])
x = np.array([[0.7],[0.3]])
sim(Q,x)
```

從markov模組匯入sim

使用sim

markov.py

```python
import numpy as np
from numpy.linalg import norm
def sim(P,x):
loop_max = 1000
loop = 0
while not norm(P@x-x) < pow(10,-6):
    x = P@x
    loop += 1
    if loop > loop_max:
        break
print(x)
```

demo_markov2.py

```python
import numpy as np
from markov import sim

Q = np.array([[0.6, 0.13],[0.4,0.87]])
x = np.array([[0.7],[0.3]])
sim(Q,x)
```

從markov模組匯入sim

使用sim

```
 [[0.24528423]
  [0.75471577]]
```

Such a file is called a module;

definitions from a module can be imported into other modules or into the main module

A module is a file containing Python definitions and statements.

The file name is the module name with the suffix .py appended.

fib模組

fibo.py

定義名稱 fib，列印小於n的費氏數列

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end = ' ')
        a, b = b, a+b
```

```
Python Console
>>> from fibo import fib
>>> fib(20)
0 1 1 2 3 5 8 13
>>>
```

從fibo模組匯入
fib定義或方法

Decimal2Binary module

```python
import random
N = 100000
S = list(range(1,N))
n = random.choice(S)
print(bin(n))
print("converting",n," to")
b = ""
while n > 0:
    r =  n % 2
    b = str(r) + b
    n = (n - r) // 2
print("0b"+b)
```

```python
import random
def  dec2bin(n):
    b = ""
    while n > 0:
        r =  n % 2
        b = str(r) + b
        n = (n - r) // 2
    print("0b"+b)
    return "0b"+b
```

回傳輸出參數

```
import random
def  dec2bin(n):
    b = ""
    while n > 0:
        r =  n % 2
        b = str(r) + b
        n = (n - r) // 2
    print("0b"+b)
    return "0b"+b
```

```
N = 100000
S = list(range(1,N))
n = random.choice(S)
print(bin(n))
print("converting",n," to", dec2bin(n))
```

最小質因數模組

輸入：一正整數n

輸出：最小質因數a

```
def least_prime(n):
    for a in range(2,n):
        if n % a == 0:
            print(a,  ' is a prime factor ')
            break
    else:
        print(n, 'is a prime number’)
        a = n
return a
```

執行for迴圈命令，
當執行到break時，
迴圈會中斷執行，
跳開for迴圈，也不
執行else:中的指令

回傳最小質因數

max_p:3

Prime: 2,3

計算步驟

1. 輸入n，設定prime為空串列，max_p = 0
2. while n > max_p:
   A. a= least_prime(n)
   B. n = n // a
   C. 如果a不在串列prime中，將a附加在串列prime中, 且將max_p設為a

輸入n，設定prime為空串列，max_p = 1

```
import random
from least_prime import least_prime
S = list(range(1,100000))
n = random.choice(S)
prime = []
max_p = 1
while n > max_p:
    a = least_prime(n)
    n = n // a
    if not a in prime:
        prime.append(a)
        max_p = a
print(prime)
```

找n的最小質因數

B. n = n // a

C. 如果a不在串列prime中，將a附加在串列prime中, 且將max_p設為a

定義**grading**，將分數轉換為等第
輸入:介於**0**到**100**之間的正整數，**score**
輸出：**g**為$'A+','A','A-','B+','B','B-'$的字串

grading.py

```
def grading(score):
    if score >= 95:
        g = 'A+'
    elif score >= 90:
        g = 'A'
    elif score >= 85:
        g = 'A-'
    elif score >= 80:
        g = 'B+'
    elif score >= 75:
        g = 'B'
    else:
        g = 'B-'
    return g
```

```python
import random
S = list(range(60,101))
N = 20
scores = []
for i in range(N):
    s = random.choice(S)
    scores.append(s)

grade = []
for score in scores:
    if score >= 95:
        g = 'A+'
    elif score >= 90:
        g = 'A'
    elif score >= 85:
        g = 'A-'
    elif score >= 80:
        g = 'B+'
    elif score >= 75:
        g = 'B'
    else:
        g = 'B-'
    grade.append(g)
for i in range(len(grade)):
    print(scores[i],grade[i])
```

```
68 B-
85 A-
69 B-
75 B
88 A-
83 B+
81 B+
75 B
98 A+
65 B-
96 A+
90 A
98 A+
83 B+
79 B
67 B-
86 A-
74 B-
88 A-
65 B-
```

95-100分為'A+'

90-94分為 'A'

85-89為'A-'

80-84為'B+'

75-79為B

其他為'B-'

```python
import random
S = list(range(60,101))
N = 20
scores = []
for i in range(N):
    s = random.choice(S)
    scores.append(s)

grade = []
for score in scores:
    if score >= 95:
        g = 'A+'
    elif score >= 90:
        g = 'A'
    elif score >= 85:
        g = 'A-'
    elif score >= 80:
        g = 'B+'
    elif score >= 75:
        g = 'B'
    else:
        g = 'B-'
    grade.append(g)
for i in range(len(grade)):
    print(scores[i],grade[i])
```

依序將串列scores中的元素代入score，執行迴圈命令

如果分數大於等於95分，為'A+'
否則如果，分數大於等於90分，為 'A'
否則如果，分數大於等於85，為'A-'
否則如果，分數大於等於80，為'B+'
否則如果，分數大於等於75，為B
其他為'B-'

將g附加在串列grade中

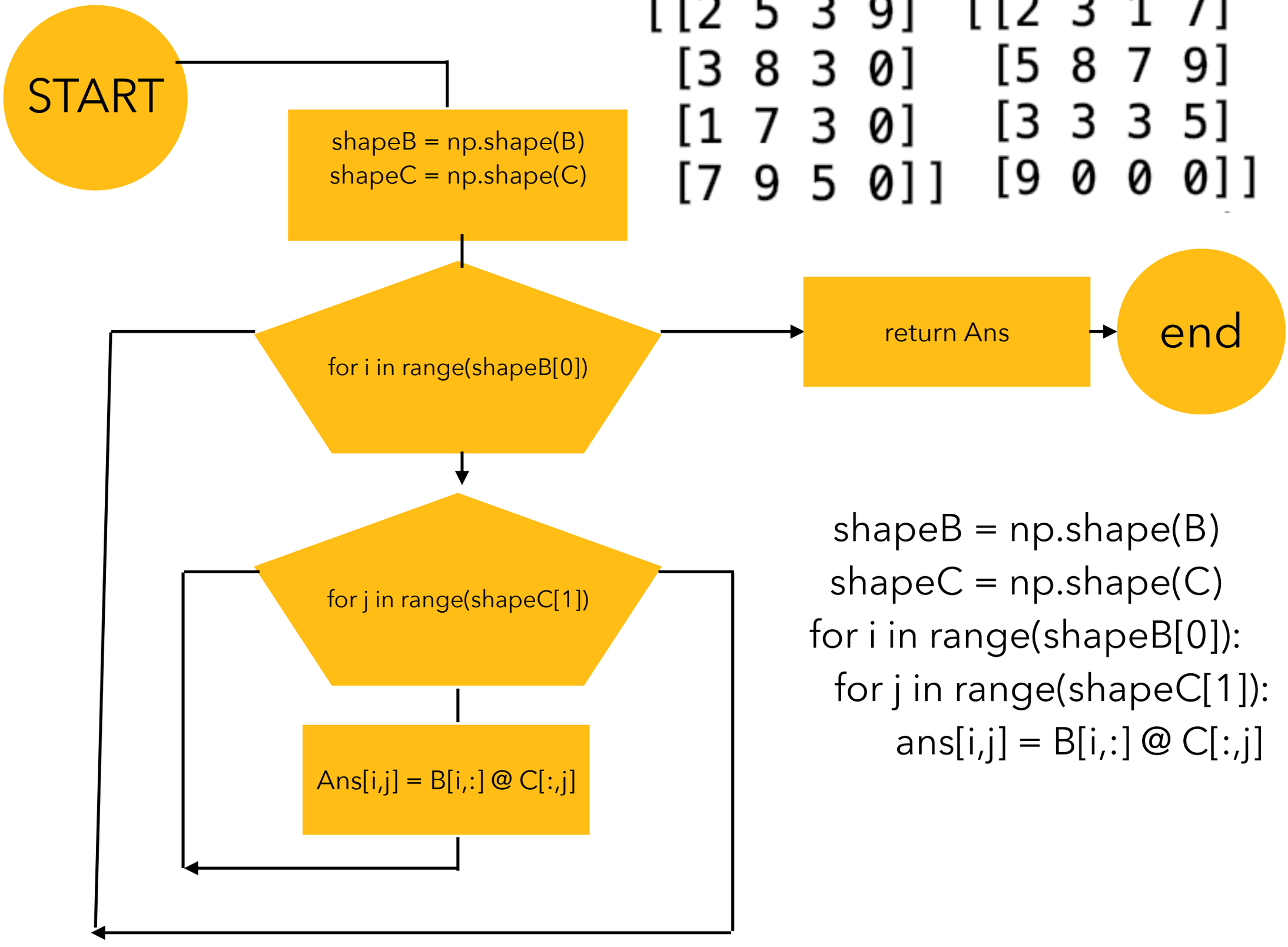
```
import random
from grading import grading

S = list(range(60,101))
N = 20
scores = []
for i in range(N):
    s = random.choice(S)
    scores.append(s)

grade = []
for score in scores:
    g = grading(score)
    grade.append(g)
for i in range(len(grade)):
    print(scores[i],grade[i])
```

demo_matrixMultiply

import numpy as np

def matrixMultiply(B,C):

```
[[2 5 3 9]  [[2 3 1 7]
 [3 8 3 0]   [5 8 7 9]
 [1 7 3 0]   [3 3 3 5]
 [7 9 5 0]]  [9 0 0 0]]
```




```
shapeB = np.shape(B)
shapeC = np.shape(C)
for i in range(shapeB[0]):
    for j in range(shapeC[1]):
        ans[i,j] = B[i,:] @ C[:,j]
```

```
import numpy as np
def matrixMultiply(B,C):
    shapeB = np.shape(B)
    shapeC = np.shape(C)
    ans = np.zeros((shapeB[0],shapeC[1]))
    for i in range(shapeB[0]):
        for j in range(shapeC[1]):
            ans[i, j] = B[i, :] @ C[:, j]
    return ans
B = np.matrix([[2,5,3,9],[3,8,3,0],[1,7,3,0],[7,9,5,0]])
C = np.matrix([[2,3,1,7],[5,8,7,9],[3,3,3,5],[9,0,0,0]])
print(B@C)
print(matrixMultiply(B,C))
```

Generate a Sudoku

Check Sudoku validity

The algorithm below will generate a NxN random sudoku solution board instantly for N < 1000.

```
base  = 3
side  = base*base

# pattern for a baseline valid solution
def pattern(r,c): return (base*(r%base)+r//base+c)%side

# randomize rows, columns and numbers (of valid base pattern)
from random import sample
def shuffle(s): return sample(s,len(s))
rBase = range(base)
rows  = [ g*base + r for g in shuffle(rBase) for r in shuffle(rBas
cols  = [ g*base + c for g in shuffle(rBase) for c in shuffle(rBas
nums  = shuffle(range(1,base*base+1))

# produce board using randomized baseline pattern
board = [ [nums[pattern(r,c)] for c in cols] for r in rows ]

for line in board: print(line)

[6, 2, 5, 8, 4, 3, 7, 9, 1]
[7, 9, 1, 2, 6, 5, 4, 8, 3]
[4, 8, 3, 9, 7, 1, 6, 2, 5]
[8, 1, 4, 5, 9, 7, 2, 3, 6]
[2, 3, 6, 1, 8, 4, 9, 5, 7]
[9, 5, 7, 3, 2, 6, 8, 1, 4]
[5, 6, 9, 4, 3, 2, 1, 7, 8]
[3, 4, 2, 7, 1, 8, 5, 6, 9]
[1, 7, 8, 6, 5, 9, 3, 4, 2]
```

2. Row check: Each row contains nine different digits

```
[1, 8, 4, 5, 3, 2, 6, 9, 7]
[7, 9, 6, 8, 1, 4, 2, 5, 3]
[3, 5, 2, 9, 7, 6, 4, 8, 1]
[8, 6, 7, 4, 5, 1, 3, 2, 9]
[5, 4, 1, 2, 9, 3, 7, 6, 8]
[9, 2, 3, 6, 8, 7, 1, 4, 5]
[2, 1, 5, 3, 6, 9, 8, 7, 4]
[6, 3, 9, 7, 4, 8, 5, 1, 2]
[4, 7, 8, 1, 2, 5, 9, 3, 6]
```

3.Column check: Each column contains nine different digits

1. Block check: Each block contains 9 different digits. There are totally nine blocks